**Abstract**

**Improving Blob Detection of Vehicles with Genetic Algorithms**

|  |  |
| --- | --- |
| Arron Bao B. Van  George Mason University  4400 University Dr, Fairfax, VA 22030  bvan@masonlive.gmu.edu | Zhaozhuo Li  George Mason University  4400 University Dr, Fairfax, VA 22030  zli9@masonlive.gmu.edu |

*We implemented a simple way of detecting vehicles from a stationary traffic camera using background subtraction and simple blob detection. Our goal at first was to detect cars on the road using OpenCV 3.0 and Python with no regards to how well it was doing. However, we mastered the basics of OpenCV 3.0 quickly and completed our assignment within a few hours. We decided to improve the accuracy without deviating from our current implementation. The genetic algorithm maximizes our accuracy by picking two instances with the highest fitness and using them to generate the entire population for the next iteration. Even with just ten generations, our vehicle detector showed great improvements.*

**Keywords**

Computer Vision, Background subtraction, Blob Detection, Vehicle Detection, Genetic Algorithm, fitness, selection, crossover, mutation, precision, recall, OpenCV, Python

**1. Introduction**

The idea of using background subtraction and blob detection are very common when it comes to detecting moving objects in a stationary background. Our objective is not to improve vehicle detection in general; there are plenty of better methods than simple blob detection. We want to improve the simple blob detector from OpenCV 3.0 by changing the default parameters. The blob detector has nineteen different parameter settings, which can be changed at any given situation. We picked five that we assumed would have the largest influence on how well the detector is doing. We also added two more settings from OpenCV Gaussian blur, filter size and deviation. The blurring helps to rid of the noise but also erases potentially important information. Our hope is that the genetic algorithm will pick the right balance between each parameter and maximize the fitness function.

**2. Related Work**

Because Computer Vision has been a very active field, numerous amounts of work were done on vehicle detection with even better methods than just blob detection.

Betke, Haritaoglu, and Davis [2] use a real-time system of vehicle detection and tracking from a moving vehicle by taking the edges, color, and template matching of cars from data taken from the internet. The system recognizes roads and the motions of other vehicles under difficult visual circumstances.

Achler and Trivedi [3] detect vehicles using the most common feature among all vehicles, the wheel. The detector was trained on a series of images of wheels and non-wheels by convolving a filter bank. By just using information from the wheels, the type of the vehicle can also be attained.

Collado, Hilario, Escalera, and Armingol [1] detect and track other vehicles using their shape, symmetry, and shadow from a camera mounted on a moving car. The system uses a genetic algorithm to find the best parameters. The system is trained using feature detectors and 3D model parameters in a neural network.

**3. Solution**

The background subtractor we chose uses density estimation adapted from Zivkovic and Heijden [5]. It works best when foreground pixel count is low. The resulting image is blurred and put into a blob detector provided by OpenCV. Every fifth pixel on the column and row is compared with the ground truth. To match with the angle of traffic cameras pointing at a road, we picked the 2014 highway dataset with 1700 frames of ground truth at http://www.changedetection.net/.

To improve performance, we coded a genetic algorithm that takes seven parameters used by the blob detector and Gaussian blur, and then maximized the fitness score by generating the population with only the top two instances of the previous generation.

Figure 1: Vehicle detector during its sixth generation, note the higher score.

Figure 1: Vehicle detector during its first generation.

**4. Experiment**

**4.1 Procedure**

We tested several background subtractors provided by OpenCV first and the one using K-NN was less susceptible to small increments in the background than others were. The camera from the highway dataset moves a few pixels to the left, enough for the entire frame to be labeled as the foreground.

Gaussian blur is applied to rid of additional noise leftover. We tested with a few filter sizes and deviations until we decided to add it into the genetic algorithm.

The blob detector then runs to try to find black blobs in the image. It returns a set of center points and diameters, which we used to draw a bounding box. Because the blob detector looks for spherical objects, the bounding boxes we use are squares.

K-nearest neighbor background subtraction from OpenCV uses the balloon estimate, which generalizes into the k-NN algorithm with certain constant factors. Instead of using fixed density estimation function on *n* number of frames to classify each pixel into foreground or background, the balloon estimate varies its width on differently populated areas of pixel intensity. This supersedes previous background subtraction methods, which used the Gaussian probabilistic density function with a fixed width.

Since the ground truth data provided no bounding boxes, we had to compare our predictions with the ground truth by pixel value. To save time, we compared pixels on every fifth row and column. The score of the detector, or the fitness function, is decided by sum of the precision and recall. The fitness function maximizes both precision and recall because it penalizes false detections and misses. True negatives were not used because the number was overwhelmingly large.

Initially, two parents are randomly generated to generate more children for the initial population. The score is calculated for each instance and the top two are picked as the parents for the next generation. To save time and utilize more computing power, we test three instances at the same time with threading in Python. The two parents are carried into the next generation along with the new population.

Crossover randomly takes 50% of the second parent and replaces it in the first parent. This generates one child and the process is repeated until the maximum population amount is reached. Children generated by parents have a chance of mutation, which helps to avoid the trouble of converging on a local maximum. Since our genetic algorithm runs at less than satisfactory speeds, we give the children a high chance of mutation in hopes of them winning the genetic lottery.

**4.2 Results**

Figure 3: Score of vehicle detector over ten generations. Maximum score is two.

The figure above is a run with ten generations and a maximum population of eight including the surviving two parents. With just ten iterations, the average precision and recall went from around 55% to 70%. The run took 21 minutes on a 3.30GHz quad-core CPU with Python only utilizing one core. Looping over pixels to match with the ground truth took a majority of the time. In conclusion, the genetic algorithm used to improve blob detection works surprisingly well. Because of the generality of blob detectors, this does not just apply to vehicle detection alone. The use of genetic algorithms can increase the performance of the simple blob detector in OpenCV in any application.

**5. Conclusion**

Further improvements can be made. Out of the nineteen possible parameters for blob detection, we only used five. Out of all the possible background subtractions, we only used one. If we were to train on bounding boxes for the ground truth, our genetic algorithm would perform must faster because the most time consuming task was matching the ground truths. The slow speed also limits our population generation. With fewer children, there are fewer chances of positive genetic mutations. More parents could be chosen with a bigger population pool to generate a diverse generation.

Another issue is that the algorithm assumes that the video is both stable and has ample lighting to allow accurate detection. In example cases where videos had bad weather, unstable/shaking camera, or low-lit area existed, the algorithm wasn’t able to accurately detect vehicles and sometimes resulted in lots of false negative detection. Handling these different use cases will require more calculations, but may improve detection under these conditions.

Increasing the number of cores used can reduce the time by allowing more than three concurrent detectors running at the same time. Perhaps a recode in C++ is necessary if performance becomes the main issue.

**References**

|  |  |
| --- | --- |
| [1] | J. Collado, C. Hilario, A. Escalera, J. Armingol. Model based vehicle detection for intelligent vehicles. 2004 IEEE Intelligent Vehicles Symposium, 572-577, 2004. |
| [2] | M. Betke, E. Haritaoglu, L. Davis. Real-time multiple vehicle detection and tracking from a moving vehicle. Machine Vision and Applications, 12(2):69-83, 2000. |
| [3] | O. Achler, M. Trivedi. Camera based vehicle detection, tracking, and wheel baseline estimation approach. The 7th International IEEE Conference on Intelligent Transportation Systems, 2004. |
| [4] | Y. Wang, P. Jodoin, F. Porikli, J. Konrad, Y. Benezeth, P. Ishwar. CDnet 2014: An Expanded Change Detection Benchmark Dataset. In Proc. IEEE Workshop on Change Detection, 387-394, 2014. Supplied as additional material. |
| [5]  [6] | Z. Zivkovic, F. Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. Pattern recognition letters, 27(7):773-780, 2006.  S. Priyadharshini, S. Dhanalakshmi. Foreground object motion detection by background subtraction and signaling using GSM, 27-28 2014. |